Programação III - Trabalho Prático

Autor: José Barroso nº42359

17 de janeiro de 2025

1. Introdução

Este projeto consiste na implementação de um programa que tendo em conta o jogo do ***desminador***, em que se pretende identificar, num tabuleiro de NxN casas, aquelas que têm uma mina. O programa escrito representa a parte "operacional" do jogo, e a interação é feita por texto. O programa irá ler uma sequência de linhas (do standard input) que representam instruções e, para cada uma, dar a resposta adequada, linha-a-linha (no standard output).

Para este problema foi utilizada a linguagem **OCaml**(Programação Funcional) uma vez que esta linguagem(em comparação com **Prolog**) tem um maior grau de semelhança relativamente às linguagens de programação mais comuns, com as quais tenho maior familiarização.

2. Implementação

Para resolver o problema em questão, é necessário ter em conta a tabela de comandos proposta no enunciado, gerando um tabuleiro e insertando as minas de modo aleatório ou modo manual, e de seguida, verificar cada casa no tabuleiro com os mesmos comandos. Sendo, o objetivo final devolver a mensagem final quando todas as minas foram marcadas com o comando mark.

Assim, para apresentar o resultado final pretendido, foram implementadas várias funções:

- **initialize_board n** – cria um tabuleiro (n x n) vazio preenchido com células vazias;
- **valid_coords size r c** – verifica se as coordenadas fornecidas (**r, c**) estão dentro dos limites do tabuleiro, garante que a linha **r** e a coluna **c** não são negativas e são inferiores ao tamanho do tabuleiro, utilizada para evitar o acesso fora dos limites;

- **count_neighbors board size r c** – conta o número de minas nas 8 células vizinhas de (**r, c**), itera por todas as 8 direções possíveis em torno de uma célula (para cima, para baixo, diagonais), utiliza a função **valid_coords** para garantir que os vizinhos estão dentro dos limites e incrementa um contador se um vizinho contiver uma mina;
- **add_mine state r c –** adiciona uma mina na posição (**r, c**) no tabuleiro, atualiza a célula do quadro em (**r, c**) para **Mine**, adiciona (**r, c**) à lista **mine_positions** para rastrear todos os locais das minas;
- **place_random_mines state k –** Coloca **k** minas aleatoriamente no tabuleiro, seleciona aleatoriamente uma célula no tabuleiro se a célula ainda não for uma **Mine**, coloca uma mina e diminui a contagem restante, garantindo que não são colocadas mais de **k** minas;
- **command_empty state n –** inicializa um tabuleiro (n x n) vazio, reinicia o estado do jogo criando um novo tabuleiro, define o tamanho do tabuleiro e limpa todas as minas colocadas anteriormente;
- **command_random state k –** coloca **k** minas aleatórias no tabuleiro, chama a função **place_random_mines** para manipular o posicionamento da mina;
- **command_mine state r c –** coloca uma mina numa posição específica (**r, c**), verifica se a posição é válida e vazia, coloca a mina e incrementa a contagem de minas;
- **command_step state r c –** simula pisar a célula (**r, c**), se a célula contiver uma mina, o output será "**boom**", se a célula for do tipo **Empty**, calcula o número de minas vizinhas utilizando **count_neighbors** e marca a célula como **Stepped n,** o output será "**count N**" onde **N** é o número de minas na vizinhança;
- **command_mark state r c –** marca uma célula (**r, c**) como contendo uma mina, atualiza a célula em (**r, c**) para **MarkedMine;**
- **command_done state –** termina o jogo e verifica se todas as minas estão corretamente marcadas, verifica se cada mina em **mine_positions** está marcada como **MarkedMine.** O output será **"ok"** se todas as minas estiverem correctamente marcadas ou **"fail"** se alguma mina não estiver marcada ou se uma célula não-mina estiver marcada;

- **command_dump state –** apresenta o estado atual do tabuleiro, imprime o tabuleiro linha a linha, entre parênteses ( ). Representado pelas seguinte caracteres, ' '(espaço) para células vazias, **'#'** para **MarkedMine**, e números de 1 a 8 para células do tipo **Stepped n** (mostrando minas vizinhas);
- **minesweeper ( ) –** a função principal que executa o loop do jogo, inicializa o estado do jogo, lê continuamente os comandos do utilizador (**empty**, **random**, **step**, **mark**, **etc.**), executa os comandos correspondentes e atualiza o estado, para quando o comando **done** é invocado ou o jogo termina;

3. Limitações do programa

Este programa apresenta algumas limitações de funcionamento, tais como:

- O jogador possui tentativas ilimitadas para encontrar as minas colocadas no tabuleiro, uma vez que quando o jogador utiliza o comando **step** e o output for **"boom"** o jogador nesse momento garantidamente conhece a localização da mina e pode usar o comando **mark** para seu benefício sem nenhum prejuízo para completar a partida, pois no jogo original ao pisar uma mina a partida terminaria;

4. Conclusão

Por fim, o desenvolvimento deste programa permitiu colocar em prática várias funcionalidades da Programação Funcional, mais especificamente em **OCaml**, principalmente no que diz respeito à manipulação de listas e do seu conteúdo, tendo sido atingido o objetivo deste projeto, simular um jogo do **desminador**, permitindo o jogador realizar uma partida desse jogo com interação feita por texto, via standard input, representadas por instruções e reproduzir o output adequado para finalizar o jogo. Ainda que com as limitações já apresentadas, o programa consegue resolver uma partida do jogo, tendo sido o objetivo deste projeto cumprido.